A MODEL MANAGEMENT SYSTEM FOR MATHEMATICAL PROGRAMS

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

By
B V NARENDRA KUMAR

to the

INDUSTRIAL AND MANAGEMENT ENGINEERING PROGRAMME
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR
FEBRUARY, 1985

CERTIFICATE

This is to certify that the work entitled, "A MODEL MANAGEMENT SYSTEM FOR MATHEMATICAL PROGRAMS," done by Shri B.V. Narendra Kumar has been carried out under my supervision and has not been submitted elsewhere for a degree.

(S. Sadagopan)
Assistant Professor
Industrial and Management Engg.
Indian Institute of Technology,
Kanpur 208016

February, 1985

ACKNOWLEDGEMENTS

It is with great respect and immense pleasure that I express my heartfelt thanks and gratitude to Dr. S.Sadagopan, Assistant Professor, IME Programme for his invaluable guidance ane cncouragement. In simple terms, he had been the lamp post for the "LAMP" - my thesis work.

I also express my wholehearted thanks to all members of IME family, for their constant inspiration and ever-helping gesture.

Finally, I want to express my thanks to Mr. V.V.S.N.Murthy, Mr. V. Jacob Neal for their help in giving the final touches to "LAMP". I also thank all my friends who made my stay at IIT pleasant, memorable and enjoyable.

(B.V. Narendra Kumar)

CONTENTS

ABSTRACT

It is without question that over the last 25 years or so most of the literature in the field of mathematical programmihg has concentrated on and contributed to the development of more efficient solution algorithms. This development together with the tremendous increase in capability of general purpose computer systems enables one to solve at present nonlinear programming involving hundreds of equations, linear programming problems havin thousands of equations, and network problems involving tens of thousands of equations.

Models have become very effective tools in planning and decision making in a large variety of disciplines and institutions and eighty to ninety percent of total resources currently spent on large modeling exercises is for the generation, manipulation and reporting of these models. The idea of reducing this percentage and treating MODELS as valuable resources got shaped into MODEL MANAGEMENT SYSTEMS.

Modelers build the model in algebraic form. But a computer needs the model to be in a matrix form. Converting model form algebraic form to matrix form is a very tedious job. To provide the user with a natural and expressive way to build a model which can readily be accepted by a computer, 'MODELING LANGUAGES' emerged.

This work is developed with the prime intention to aid a decision maker in building Linear Programming models in a much more convenient way.

The basic features of this system are:

(i) Language Aided Mathematical Programming (LAMP) is a PASCAL based language.

(ii) This system consists of three-basic modules: TERMINOLOGY, DATA BASE and ABSTRACT MODEL. These three systems form basis to the rest in building up the model.

(iii) Syntax of LAMP is deliberately kept close to algebraic notation to help the user.

(iv) Apart from the direct input of the data by the user, LAMP provides a special facility of inputting the DATA from external files. Especially for large problems, user is relieved from inputting the large data.

(v) LAMP accepts english like declaration of abstract model and reports the results back in algebraic form.

In a nutshell, LAMP acts as mediator between decision maker/modeler and the computer system. Systems like LAMP should aid the modeler in building models easily and more efficiently.

CHAPTER I

INTRODUCTION

1.1 INTRODUCTION:

In the early days of mathematical modeling, large applications were mostly of a military and industrial nature. Models were used to describe and solve well defined problems in the areas of production and distribution, and they were employed on a routine basis. In many instances it was considered cost-effective to establish a small group of technical people whose sole responsibility was to maintain and to improve the existing package of models. In recent years the scope of mathematical modeling applications has widened, and modeling environments different from those described above has emerged.

In the policy/planning environment the role of models is often extended beyond their traditional use as a way to get numerical solutions to well-defined problems. Models are used to express perceptions and abstractions of reality, and they continuously change as their developers learn more about the uncertain real-world problem. Models provide the model builder/decision maker with a formal frame work for data collection and analysis. They are used to simulate, predict or plan the behaviour of physical or social systems [4].

1.2 NEED FOR MODELING SYSTEMS:

In the early stages the size and complexity of models were severely limited by technical and budgetary constraints. In fact, models remained so small that algebraic formulation and data could be written down as a few sheets of paper [6].

A phenomenal improvement in computing power and algorithmic capability has entirely changed the picture. Spurred by the evolution of sophisticated computers, the scope of the mathematical modeling applications has widened, and secured a formidable base in almost all fields. As models have grown in size, an capability to comprehend and control them has diminished rapidly.

1.2.1 Problems in Large-Scale Modeling:

Generally as the model grows in size, problems arise. They are identified as follows [4]:

(i) The documentation of large models and their modifications is very difficult. If a project continues for one or two years, the cost of complete documentation becomes horrendous.

(ii) Communication of models to interested persons outside the project term becomes another related problem. As there are no standards in notation, it is often difficult to judge from any write-up what exactly the model is.

(iii) Experimentation with the models may enhance ones understanding, but this requires the use of latest technology available (i.e. report generators). The extensive time and money requirement prohibit the effective dissemination of knowledge to outsiders who are curious about the model.

1.2.2 Computerised Modeling Systems - A Remedy.

Although the above problems tend to discourage large-scale modeling exercises, they are not major obstacles to theeffective use of modeling in a policy/planning environment due to computers. Here we gradually move away from the existing labour/skill intensive approach to model building, and replace it with a machine intensive approach.

The evolution of theuse of computers in moder organisations has led from transaction oriented data processing systems to report oriented information management systems and finally to interactive decision support systems. Development in the management science and information systems fields have led to the present recognition of the importance of incorporating extensive data handling capabilities and models into a single system with which decision makers can directly communicate. Thus the trend has been directed away from fragmented views of decision support, towards a workable integration of the two decisions support functions of data handling and modeling [7].

As more people in decision making position around the world are becoming interested in applying mathematical programming techniques, large number of software systems are developed to aid them in decision making.

Among these systems, some are model oriented while some are data oriented, some provide flexible user languages, while others offer languages with strict command format and syntactic rule.

To help the decision maker in his decision making process, a system should be [5]

(i) Generalised (it can be used to design a decision support system in any application area).

(ii) Powerful (it supplies facilities for modeling, model management, data handling and linking the model and its data) and,

(iii) Friendly (it provides an easy-to-use language to facilitate user-system interaction).

Though until now no system could meet the above requirements fully, efforts to satisfy them has given birth to the concept of MODEL MANAGEMENT SYSTEMS.

The development of generalised DSS (Decision Support Systems) software is a complicated task. Such a system should not be developed by simply patching together a group of stand-alone software components. Neither should it be just an

extension of another software package by adding some Decision Support Systems features. Although Decision Support Systems often emphasize modeling capability, data handling capabilities cannot be neglected. Moreover, many users of Decision Support Systems are relatively naive , at least with regard to some dimensions of the decision process. In order to develop a generalised, powerful and friendly Decision Support Systems software system, the modeling and data handling capabilities and the user-system interface must be well-designed from top down and especially adaptive to differing degrees of user expertise. Model Management System had its origin from these considerations and formed the basis for various DSS software systems.

Among the various models available in mathematical programming, linear programming is a valuable model especially in areas like production planning and scheduling. Linear programming has found its ways into almost all areas of application. This wide acceptance and usage of Linear programming became basis for evolution of many software systems. We also concentrated to develop a software system for Linear programming models considering its wide applicability and algorithmic capability.

These efforts finally took a form by the name LAMP (Language Aided Mathematical Programming).

1.3 OUTLINE OF THE THESIS.

The main aim of this thesis work is to design and implement user interface system for solving linear programming models. This thesis presentation covers the fundamentals of Model Management Systems (Chapter II) and explains the system LAMP (Chapter III).

Chapter II mainly focusses on the conceptual model and operational environments for Decision Support Systems. This chapter also emphasizes the idea that Data Base Management Systems and Model Base Management Systems are parallel. A survey of available modeling systems is also given.

Chapter III puts-forth the actual work done. This chapter explains the design and implementation of LAMP and also provides a typical Linear Programming Model (Production Scheduling) emphasizing the algebraic form, matrix form and modeling through LAMP.

Chapter IV gives an idea of the possible extensions that can be incorporated in LAMP.

CHAPTER II

MODEL MANAGEMENT SYSTEMS

2.1 INTRODUCTION:

Model Management and Model Management Systems (MMS) are relatively new concepts which have emerged from the recent upsurge of interest in Decision Support Systems (DSS). Just as the Data Management approach highlights the need for treating the data as a valuable resource to be managed systematically by the organisation, the Decision Support System philosophy argues for an equivalent approach to models. Models are instruments which transform data into information which can aid decision making.

2.1.1 Conceptual Model for Decision Support Systems:

Generally a powerful Decision Support System software supplies facilities for modeling, model managing, data handling, linking models and data and user-system interaction [7].

Model Management System dynamically constructs a decision aid in response to a particular problem. The process is accomplished by drawing on a knowledge base of models that reflect the technical expertise of a management scientist and the organisation's experience, with the activities involved in a given decision making environment.

Generally a decision support system will be having three principal components (Fig. 1)[7].

(i) Language System:

Language system is referred to as the sum total of all linguistic facilities made available to the decision maker for retrieval and computations.

In the conceptual Decision Support Systems the user will be involved in decision making who makes decisions based on the models and data available to him. Generally, a Language is used to develop models, to establish dialogues that provide HELP facilities, and to create prompts for data entry.

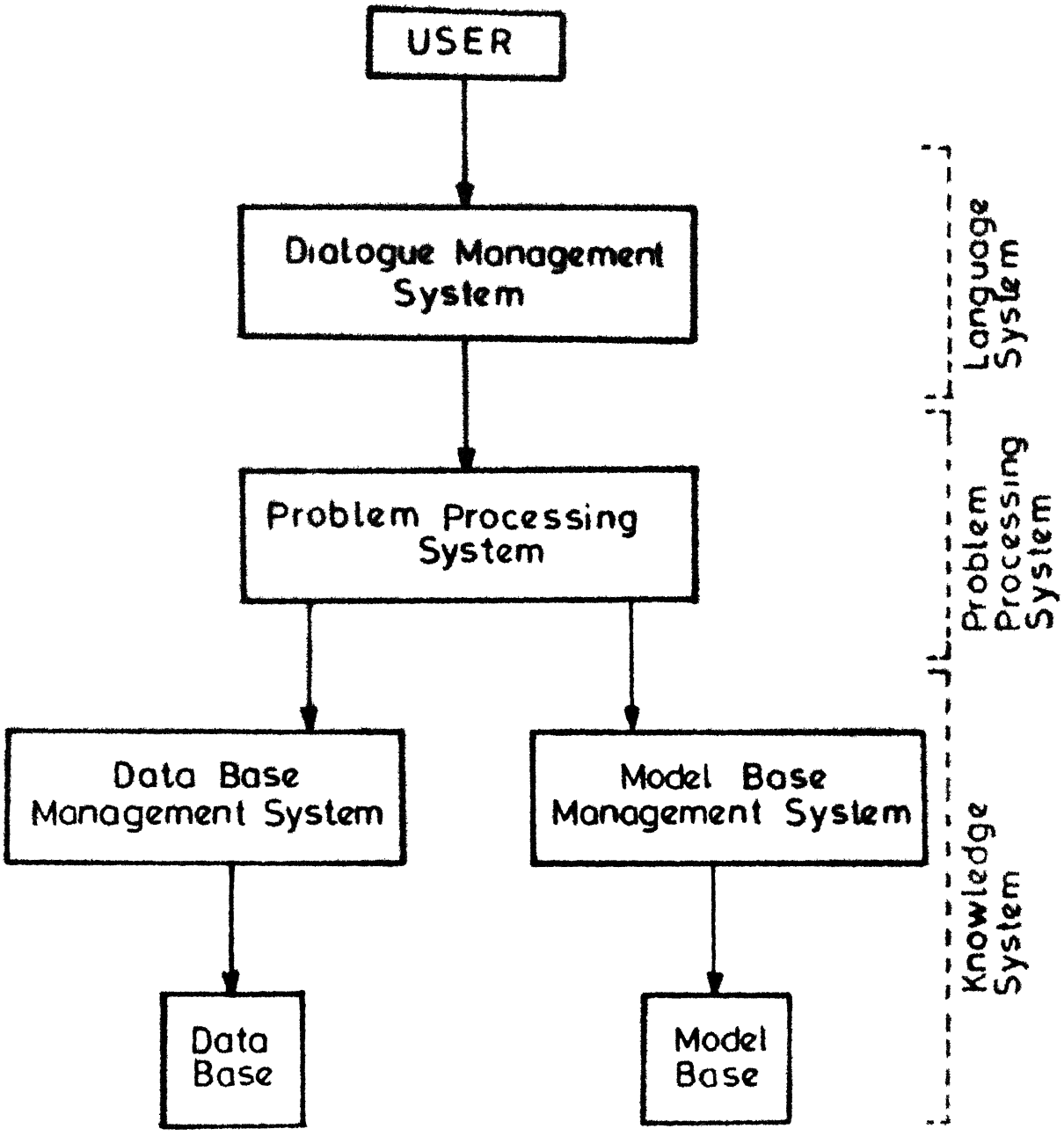
(ii) Knowledge System:

A knowledge system is referred to as a decision support system s body of knowledge regarding a problem domain. The knowledge is expressed in two ways:

(a) regarding the model that fits for the problems i.e. Model Base Management System.

(b) Data that should be fed in i.e. Data Base Management System.

These two correspondingly create a model base and a data base.



1 Conceptual Model for decision Support System

(iii) Problem Processing System:

The mediating mechanism between expressions of knowledge in the knowledge system and expressions of problems in language system is referred as problem processing system. This system uses algorithms like Linear Programming, Regression, Simulation to solve the model.

2.1.2 Operational Environment of Decision Support Systems:

Decision Support Systems are man-computer systems used by a Decision Making System on an on-going basis for supporting decision making activities (Fig. 2). A Decision Making System consists of a system administrator (Coordinator), information suppliers, model builders (Suppliers of analytical tools) and decision makers [7].

A Decision Support System consists of DSS configuration, Model Base, Data Base and Operational Procedures.

The system administrator of a Decision Making System is responsible for selecting appropriate DSS software, coordinating the decision making environment, configuring the DSS, setting up the DSS operational procedures, and monitoring the operations of the Decision Support System.

Constructing a MODEL includes providing mechanisms for handling data and/or algorithms for solving modeled problems. The model builder provides composite data-base queries and algorithms.

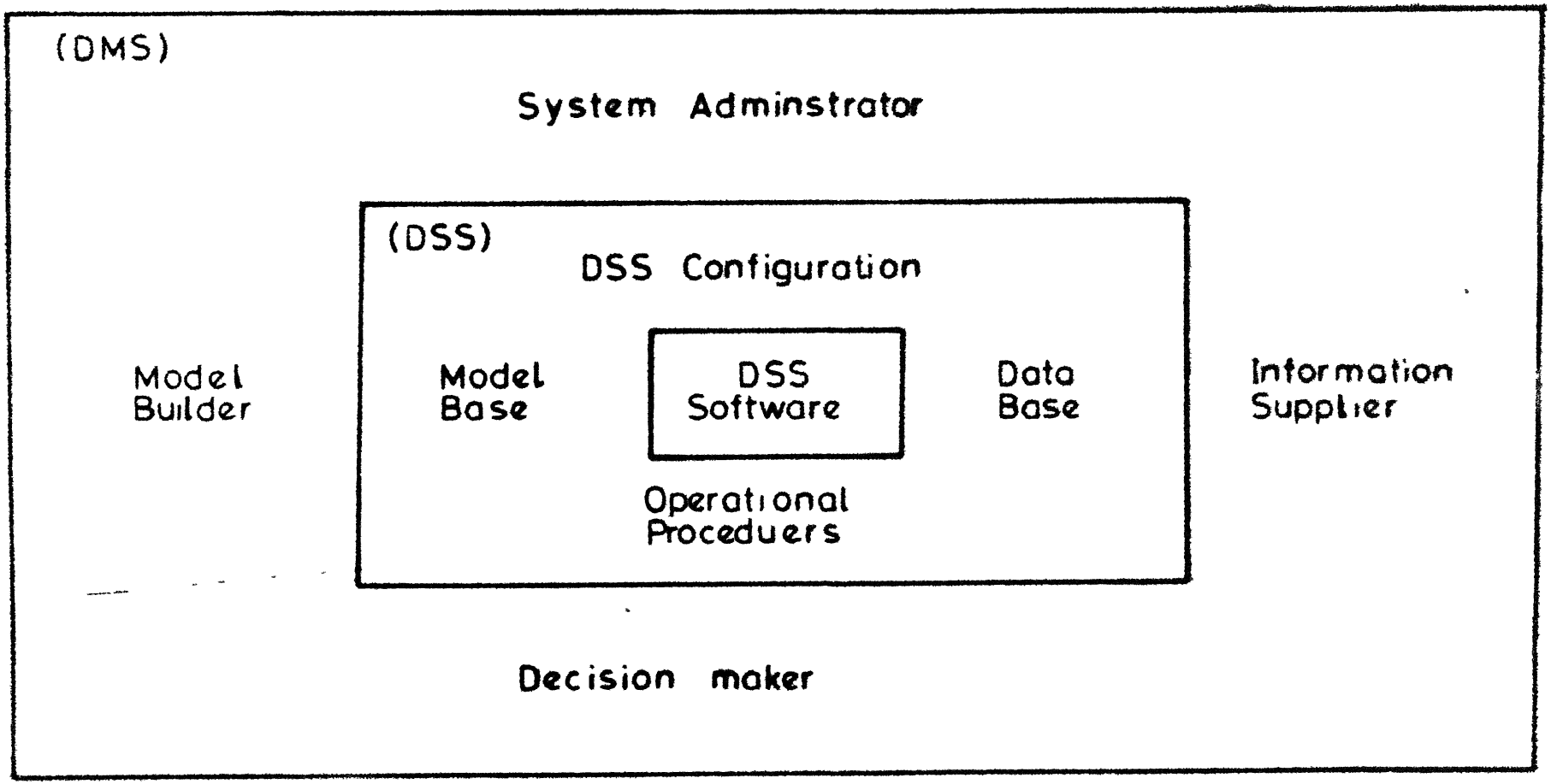


Fig. 2 The operationl environment of a decision Support System

Decision maker is responsible for handling decision making tasks and selecting appropriate algorithms.

2.2 MODEL MANAGEMENT SYSTEMS VS. DATA BASE MANAGEMENT SYSTEMS

Data Management and Data Base Management Systems (DBMS) evolved as a means of controlling data within an Organisation. The main purpose is that data should be sharable by many applications instead of each application having its own separate data files and formats [1]. This reduces the data redundancy and results in data consistency, increased accuracy of data through implementation of integrity mechanisms, the ability to enforce data presentation standards and in general a much higher degree of control over the data resource.

The arena of Models and Model Management can also be viewed in similar lines. Many organisations are now experiencing the same lack of control over model resources that were previously prevalent on data domain. Management gradually realised that models are another valuable information resource and subsequently must be managed and controlled as such.

Data Base Management System has evolved with the prime intention of providing flexible facilities to store, extract, aggregate (relate), update, and retire (destroy) data. DBMS should also be able to load data from external files into the data bases or store data in external files. DBMS provides decision makers with a logically clear view of data.

Data Base Management System also provides the Data independency both physically and logically. It allows 'tuning' of the physical data base for efficiency while permitting application programs to run as if no change had occurred (i.e. physical independency). DBMS also allows a facility to modify the conceptual scheme (i.e. logical independency) [8].

The advances in DBMS provided a foundation for designing Model Base Management Systems. In Model Base Management System modeling activity is viewed as actualising the user desires rather than merely mathematical modeling. MBMS provides a flexible way to define, invoke and delete models in the knowledge base. It also provides facilities to register and access external model-building blocks to fully utilise existing resources. In MBMS also we aim for the independence between Model and Data. A Model Base Management System should allow conceptual changes in the model and also physical changes in data without effecting the rest of the system.

2.2.1 Linkage Between MBMS and DBMS.

One of the main obstacles to model implementation is the cost of data preparation. A set of data must be restructured to be used by different models. The need for data restructuring interrupts the communication between different models [7]. By incorporating Model Base Management System and Data Base Management System in an integrated system, above problems could be

solved i.e. by handling the data and models in a consistant way. For restructuring of data, various types of data structures can be used to integrate the model building blocks with data.

2.3 FUNCTIONS OF A MODEL MANAGEMENT SYSTEM:

The purpose of a Model Management System is to make available a wide variety of models (e.g. Linear Programming, Forecasting, Regression, Simulation) to decision makers so that they can apply them to appropriate areas of application.

The functions of MMS can be stated as follows: [1]

(i) A Model Management System must be knowledge-based in order to capture the dynamic aspects of the decision maker s environment.

(ii) A Model Management System should be general enough to handle many different classes of models as well as support multiple views of the same model. Model Management System should allow a natural and convenient interface between the user and his model.

(iii) A Model Management System should be developed as much as possible as an analog of a Data Base Management System. This not only insures compatibility with existing Data Base Management System resources but also provides a rich body of principles and structures from which to draw in building MMS's.

Though there are variety of models are available, Linear Programming has emerged as a powerful aid for decision makers in almost all fields. This success is mainly due to the ease for formulation, efficiency of simplex algorithm and availability of facilities like sensitivity analysis, and parametric analysis.

Many software systems are now available to solve Linear Programming models extending various facilities to the user.

2.4 SURVEY OF MODELING SYSTEMS (LANGUAGES):

A Linear Programming modeling language could be defined by the following two requirements [2].

(1) It must be possible to express any linear program in modeler s form by use of the modeling language. A modeler s form of a Linear Programming is a notation that expresses both what the LP is and how it relates to the situation being modeled.

(2) It must be possible to create a computer system that takes any modeling language LP as input and that produces a corresponding algorithm s form of the LP as output.

Here are some languages which are capable of representing fairly large models.

ALPS: (Advanced Linear Programming System)

ALPS developed by PROSE, Inc., is being distributed by United Computing Systems. This language is fully

algebraic, uses subscripted identifiers. The model description is not independent of explicit data.

GAMS: (General Algebraic Modeling System)

GAMS is a project of the Development Research Centre of the World Bank. This language also uses subscripted identifiers for model components. It can allow General Linear Expressions and indexed Sums. Here also the data description is not independent of explicit data.

LMC: (Linear Modeling Capability)

LMC is a subsystem of CML (Conversational Modeling Language) developed at the centre for the study of Health services, Yale University. This language is more English like rather than algebraic. This also allows General Linear Expressions and indexed Sums. At times the data description here is independent of explicit data. LMC also uses subscripted identifiers.

LP MODEL: (Linear Programming Modeling Language)

LP MODEL was developed at IBM Israel Scientific Center. This language resembles common algebraic notation but altered moderately. This LP MODEL uses concatenated identifiers of unrestricted length, used like subscripts. Here data description is always independent of explicit data. LP MODEL allows general linear expressions and limited indexed Sums.

UIMP: (User Interface for Mathematical Programming)

UIMP is a product of UNICOM Consultants Ltd., and SIA, Ltd., UIMP is a Fortran-based language which is fully algebraic. UIMP allows very limited linear expressions and general indexed Sums. This language uses subscripted identifiers. In this language parameters may be independent of explicit data.

CHAPTER III

LAMP - SYSTEM DESIGN

3.1 INTRODUCTION:

The conceptual framework for organisational decision making furnishes a basis for the design of a generalised intelligent decision support system (GIDS), that will have two major parts: an information base and a generalised intelligent problem processor. They also provide three-basic systems (explained in 2.1.1): a language system, a knowledge system and a problem processing system.

Among the various models used in the Model Management Systems, Linear Programming is a powerful and valuable aid in decision making especially in the areas of production planning and scheduling. The success of Linear Programming lies in two facts. First, many and diverse practical problems require (or can be formulated as) minimisation of a Linear Combination of variables, constrained by linear equalities and inequalities. Second almost every such problem can be solved routinely and efficiently by use of a single general algorithm, the Simplex Method.

Corresponding to above two observations, there are two necessary forms of a Linear Program. When a modeler builds an LP, he expresses it in its natural "algebraic form":he defines constants and variables of theproblem, writes an objective as an arithmatic expression i.e. linear in the variables, and writes the constraints as equalities or inequalities between linear expressions. The simplex algorithm, by contrast, needs the problem in its "Matrix Form". a series of column vectors, each column being the coefficients of one variable [3]. Section 3.5 brings out the contrast between algebraic and Matrix forms.

Modelers cannot work efficiently with matrix form and the algorithm cannot employ algebraic form. As a result, computer systems for Linear Programming face three tasks:

(i) Translate the model from modeler's (algebraic) to algorithm's (matrix) form.

(ii) Solve, using the simplex method

(iii) Report the solution in modeler's (algebraic-form) terminology.

3.2 THE TRADITIONAL APPROACH:

The first concerns of Linear Programming system designers was the implementation and perfection of the simplex method. Enormous efforts were needed to collect the data and organise it as a matrix (i.e. algorithm's form). At the early stage the principles of operating systems, file systems, and interactive

computing were primitive or unknown. Hence LP systems were essentially big programs that ran as batch jobs.

3.2.1 Matrix Generators:

For problems of any size and complexity, specifying every matrix element by row and column number was hopelessly inefficient and error-prone. A more practical scheme for matrix input quickly arose: rows and columns were given unique names, typically of upto 8 letters and numbers. Each non-zero matrix element was then specified by giving a row name, a column name and a value. A common input format for this arrangement was a 'matrix deck' in some standard form as MPS.

Still, translating an algebraic - form LP to a matrix deck required much repetitive and tedious work. It was here, computer technology had its involvement. The logical next step was a "matrix generator" (MG) system designed specifically for creating matrix decks.

An MG was operated by writing a program in a specially-designed language. The program first declared sets of indices and tables of numerical data, it indicated how names of rows and columns were to be formed, and for each column, it specified that non-zero matrix elements. [3]

Matrix generators offered numerous advantages to hand-coding of matrix decks:

- They provided for organising and storing the problem data.
- They made it easy to enforce a uniform scheme for naming rows and columns.
- They led a user change model structure of model data with much less work.
- They reduced clerical errors, moreover, the logical structure of MP program made mistakes in formulation somewhat easier to catch.

Although, MG's had so many advantages they still had had some serious limitations:

- MGs still ran in batch mode and required their own file structure and environment.
- Writing an MG program was seldom easy, and was impossible till one mastered the special MG programming language.
- Large MG programs are hard to follow.

These limitations led to a new approach to LP modeling where the user is provided with a natural and easy way for inputting the data.

3.3 A MODERN APPROACH:

There are two principal aspects to what we see as a modern Linear Programming system:

First, problems are described to the modern system in algebraic form, using customary mathematical notation as much as practicable.

Second, the modern system is designed to take advantage of relatively new and powerful ways of using a computer, such as interactive operation etc.

3.3.1 Describing the Model in Algebraic Form:

Models are first written, and usually are best understood in algebraic form. Ideally, an LP system would lead the modeler's algebraic formulation directly, would interpret it, and would then generate the appropriate matrix.

This ideal is beyond the abilities of current-day computers. But a modern system can come significantly close to it - by employing a variant of algebraic form that is designed to be lead by a computer system and we call this machine-readable algebraic form a modeling language. It differs from common algebraic form mainly in employing standardised notation and terminology, and in requiring typewritten equivalents of subscripts and summation operator (Σ).

However, there are two fundamental differences between Modeling Languages and Matrix-Generator languages: [3]

First, a modeling language makes no reference to the Linear Programming Matrix. It serves only to represent an algebraic form of the model. MG languages on the other hand, are just the opposite: they describe a Linear Programming by specifying all of its non-zero matrix coefficients.

Second, a modeling language is not a computer-programming language. Rather, it is declarative. It serves only to describe a LP in a convenient way. A modern system reads this description, analyses the described Linear Program, and automatically creates the appropriate matrix. By contrast, an MG language is a programming language whose statements describe explicitly the creation of a matrix.

3.3.2 Advantages of a Modeling Language (ML):

As a consequence of above differences, a modeling language can make the LP user's life easier in number of ways compared to MG languages:

- Modeling Language is easier to learn since it looks as much as possible like common algebraic notation.
- Once a Linear Program is formulated algebraically, translating it to the Modeling Language is essentially just a job of transcription.

- Modeling Language can also serve as documentation of the Linear Programming model as it is close to algebraic description.
- As Modeling Language is easily understood, it is also not hard to find mistakes and make changes.
- Modeling Language identifies constraints and variables by the familiar method of subscripting.

To materialise all the above explained advantages, an efficient translator is a must. The main aim of this work is to design and implement such a translator which in turn becomes a part of the system LAMP.

3.4 DESIGN OF THE SYSTEM "LAMP':

Generally a Linear Programming model consists of a set of Linear constraints and a Linear objective function to be either minimised or maximised. The most basic property of a Linear Programming model is that once the problem is described, a standard procedure is adopted for its solution. Apart from these basic properties, some more observations can be made about Linear Programming models:

- Variables do generally form groups
- Some of the constraints have a similar structure
- Some aspects of the model change more often than others. In many cases one conceptual model is applied to different sets of numerical values.

In other cases some of the groups of variables or constraints are slightly changed, whereas the model as a whole retains its structure.

A careful study of the above properties and considerations led to the development of three-modules which when put together constitute the present work. These three modules are:

(i) Defining the terminology which forms a basis for the abstract model.

(ii) Creating and maintaining the Data Base that represents the known quantities in the model.

(iii) Expressing the abstract model.

3.4.1 Defining Terminology:

In order to be specific and precise about the nature of the module called TERMINOLOGY, here are few definitions:

Name: a sequence of ten or less characters with no blanks in between.

Atom: is a name which in itself represents some aspect of reality and does not stand for any other name in the model. e.g., Lathe, Cotton.

Molecule: is an abbreviation for a list of atoms.

As the decision variables of a Linear Programming model generally form groups, the names given to these groups become molecules. The names of members in each group become atoms.

In order to complete the first module, the user has to declare all the molecules and atoms. If a name appears later in the process which is not previously described then it is assumed to be an atom and will be added to the group "unidentified atoms".

3.3.2 The DATA BASE:

The data base subsystem is used to associate values with the identifiers that stand for constants in an abstract model. Here an identifier is either a single name or a series of names separated by periods.

For example: TOT-BUDGET, COST.PRODUCT,
MATERIAL.SHOP.PRODUCT

There is a canonical order among the primitive identifiers represented by an identifier. A primitive identifier comprises of only atoms. The first primitive identifier consists of the first atom from each molecule in the identifier the next one leaves all unchanged except the right most molecule (where the next molecule is used) and so forth.

Example: If the Terminology can be taken as:

Molecules	Atoms
PRODUCT	NUT, BOLT
SHOP	DRILLING, MILLING, LATHE

Then an identifier like MATERIAL.SHOP.PRODUCT would form following primitive identifiers:

MATERIAL.DRILLING.NUT, MATERIAL.MILLING.BOLT

MATERIAL.MILLING.NUT, MATERIAL.MILLING.BOLT

MATERIAL.LATHE.NUT, MATERIAL.LATHE.BOLT

The data have system gives values to those identifiers used as constants by associating a value with each identifier. If an identifier is used as a variable later in the abstract model then all the primitive identifiers are assigned with a value 1.

The system provides two choices.

One, the system when prompted appropriately (as explained later in (i) in 3.5.2) will flash the primitive identifier (formed as explained above) and waits for the data to be typed in. When the data is typed, it flashes the next one. This process is repeated until all primitive identifiers are taken care of.

Second, the data could be set from an External File (i.e. indirect inputting).

3.4.3 The ABSTRACT MODEL.

It has been already explained about the Linear Programming models in 3.4 and it could be represented as:

Maximise/Minimise $\sum_{j=1}^{n} c_j x_j$ (Objective function)

Subject to $\sum_{j=1}^{n} a_{ij} x_j \leq b_i , \quad 1 \leq i \leq m$

(Constraint Set)

$x_j \geq 0$

Here i and j are indices and they could be representing a particular shop and a product produced. All the values of i and j give rise to sets of objects or members. Analogous to these sets are Molecules in the system. The atoms represent the values of i or j under a particular set.

The abstract mathematical notation is quite precise and concise but it is not natural and easily understood by the user who may not be aware of Linear Programming all the way to its core. However, the same problem stated in ordinary English becomes more natural, expressive and understandable.

If a_{ij} is the material for j-th product in i-th shop then this can be easily written as MATERIAL.SHOP.PRODUCT in this system. Similarly if b_i is the total available material in each shop then that could be represented as TOT-MATRL.SHOP in the present system.

The "Σ" notation is replaced by the option word SUM. The index over which summation has to be done can be specified. The character "?" is chosen to follow an identifier intended as a Linear Programming variable.

A constraint like sum of the material required for all the products in a particular shop should be less than or equal to the total material available in that shop could be represented as:

SUM [PRODUCT:MATERIAL.SHOP.PRODUCTxPRODUCT?]≤TOT-MAT.SHOP

This declaration will produce one constraint for each shop.

The system also facilitates the declaration of various forms of constraints as:

(i) $\sum_{j} a_{ij} x_{j} \leq b_{i} \quad \forall\, i$

(ii) $\sum_{i} a_{ij} x_{i} \leq b_{j} \quad \forall\, j$

(iii) $\sum_{j} x_{ij} \leq b_{i} \quad \forall\, i$

(iv) $\sum_{i} x_{ij} \leq b_{j} \quad \forall\, j$

(v) $\sum_{i} \sum_{j} a_{ij} x_{j} \leq B$

(vi) $x_{j} \geq b_{j} \quad \forall\, j$

(vii) $\sum_{j} a_{j} x_{j} \leq B$

The objective could also be declared using the same syntax as that of a constraint but the word "SUM" is replaced by either "MAX" or "MIN".

The syntax is intentionally quite close to the familiar notation for arithmatic expressions used in high school algebra. As explained earlier if an identifier is used as a variable in

the constraint declaration and was not previously declared under the second module then all the premitive identifiers covered by that identifier will be assigned a value of '1'.

3.5 IMPLEMENTATION OF THE SYSTEM "LAMP":

This system was designed and implemented on DEC 1090 main frame. "LAMP" system is written in PASCAL and the system uses "LINDO" package available on DEC system as the problem processing system [Fig. 3].

"LAMP" has four subsystems. They are.

(i) Model Editor

(ii) Data Editor

(iii) Model translator

(iv) Problem solver and solution reporter

3.5.1 Model Editor:

Creates and updates model descriptions written in the Modeling language. Model Editor is implemented in three phases.

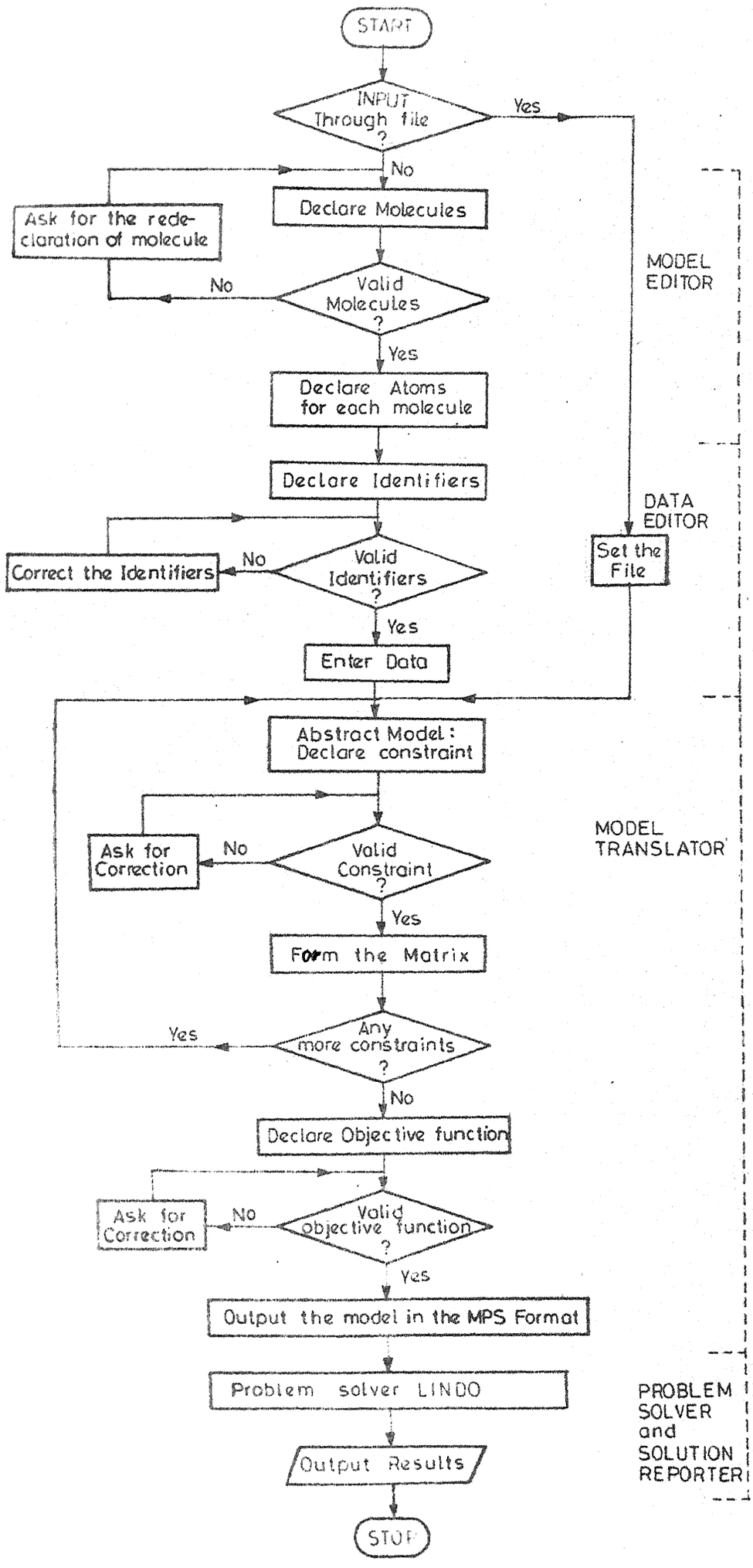
1. The system will be assisting the modeler by giving appropriate messages of commands or syntax to be followed. The system prompts the user to choose the appropriate cause of action when any one part of the declaration was over.

For example: Please type any one:

1 : for declaring Molecules

2 : for declaring Atoms

..... so on.


START
INPUT Through file ?
Yes
No
Declare Molecules
Ask for the rede-claration of molecule
No
Valid Molecules ?
Yes
MODEL EDITOR
Declare Atoms for each molecule
Declare Identifiers
DATA EDITOR
Correct the Identifiers
No
Valid Identifiers ?
Set the File
Yes
Enter Data
Abstract Model: Declare constraint
MODEL TRANSLATOR
Ask for Correction
No
Valid Constraint ?
Yes
Form the Matrix
Yes
Any more constraints ?
No
Declare Objective function
Ask for Correction
No
Valid objective function ?
Yes
Output the model in the MPS Format
Problem solver LINDO
PROBLEM SOLVER and SOLUTION REPORTER
Output Results
STOP

2. System will accept a name consisting of 10 characters or less only. If the number of characters in any name exceeds ten or if a blank appears in the name then error message is flashed.

3. Once all the molecules were declared and when the modeler prompts the system for declaring atoms, the system will ask the user to give the list of atoms under the first molecule and when the list is furnished, it automatically asks for the next. This process is repeated for all the molecules.

3.5.2 DATA Editor:

DATA EDITOR stores and updates the numerical data and index sets for models created by the model editor. Here Data Editor provides a choice to the user in inputting the data i.e. user can input the data directly in interactive mode or a file containing the data can be set to the system from which the data will be accessed. Data Editor is implemented in three phases.

1. If the modeler wants to enter DATA, then he has to type the character "$" at the end of the identifier. Then the system flashes the primitive identifiers generated by that identifier one by one and waits for their values to be typed in by the user (as explained in 3.4.2).

2. If any molecule part of the identifier (that is to be varied canonically 3.4.2) is not declared then the system flashes appropriate messages and the modeler has to type only that part.

Thus only after verification is done, the primitive identifiers are formed.

The syntax to be followed for identifier:

<atom> (or) <atom>.<molecule> (or)

<atom>.<molecule 1>.<molecule 2>

3. Inputting Data from External Files:

Inputting the data directly in the interactive mode (as explained earlier in 3.4.2 and item 1 and 2 of 3.5.2) will be convenient as far as the model is small or medium. But as the model becomes large, direct inputting will be very inconvenient and time consuming. To relieve the modeler from this, a special feature is provided in the system i.e. the data can be input from external files.

The external file can have all the molecules along with the atoms as well as the identifiers along with the values for their primitive identifiers. The system gives a choice to the user/modeler regarding the data-input. If he chooses to input through a file then he has to give the name of the file. The system reads the whole data and is now ready for the next module i.e. abstract model declaration.

Format to be followed in external files:

```
MOLECULE 1  =  atom 1, atom 2 ...
MOLECULE 2  =  atom 1, atom 2 ...
.....
*
IDENTIFIER $
{input data}
```

3.5.3 Model Translator:

Model Editor and Data Editor now form the basis for the Model Translator. The model translator is now ready to accept the abstract model of the Linear Programming. Implementation of model translator is carried out in three-phases.

1. When the modeler is in the third module i.e. in the declaration of abstract model, the system assists him at every stage showing the syntax to be followed and also flashing appropriate messages when any syntax error occurs or any part is not declared already.

2. If the identifier is used as a variable then the system in order to clarify the situation, asks the validity of the declaration and if modeler declares it valid then it gives a value 1 for all the primitive identifiers formed under that identifier.

Syntax for constraints:

1. SUM [MOLECULE: IDENTIFIER * MOLECULE?] (<=) OR (>=) OR (=) IDENTIFIER2

for $\{ \sum_j a_{ij} x_j$ (<=) OR (>=) OR (=) $b_i \; \forall \; i \}$

2. MOLECULE? (>=) OR (<=) OR (=) IDENTIFIER

for $\{ x_j$ (>=) OR (<=) OR (=) $b_j \; \forall \; j \}$

3. SUM SUM [MOLECULE: IDENTIFIER * MOLECULE ?] (<=) OR (>=) OR (=) IDENTIFIER2

for $\{ \sum_i \sum_j a_{ij} x_{ij} \leq$ OR $\geq$ OR = B $\}$

4. SUM [MOLECULE: MOLECULE?] (>=) OR (<=) OR (=) IDENTIFIER

for { $\sum_j x_j$ (>=) OR (<=) OR (=) B }

3. The system follows "CHECK AND CORRECT THEN AND THERE" logic while it parses the declarations. It immediately flashes messages about the error detected and asks the modeler to correct it then and there itself so that modeler is free from unnecessary and complicated problems afterwards.

This part of the system converts the abstract model declared into a file having MPS-format. Later this file will be the input to the problem solver and solution reporter.

3.5.4 Problem Solver and Solution Reporter:

'LAMP' uses 'LINDO' (Linear, Interactive and Discrete Optimizer) available on DEC 1090 as the problem solver. LINDO is a friendly system designed to be immediately useful to both the novice and expert to solve Linear Programming Problems.

LINDO accepts data from external files stored in MPS format. The MPS format for describing an Linear Programming is a format commonly used in industry. The model translator s output is a file stored in MPS format which will be directly fed into the LINDO which solves the problem and reports the output results.

At any point of time LINDO stores the problem formulation as well as the results of the computation. The data read from

the file will be first converted back to ordinary algebraic format and will be stored. LINDO uses the simplex algorithm to find the optimal solution for the problem under consideration and stores the results of computation. It reports the optimal solution and the iteration in which that solution is reached. It displays the values of decision variables at optimal point as well as the optimal objective function value.

3.6 A TYPICAL LINEAR PROGRAMMING MODEL (Production Scheduling):

A metal processing plant receives an order to produce 10,000 casings. The contract specifies a sales price of $5.00 per casing. The products design engineer proposes four alternative designs for the casings resulting in different variable machine time usages and material costs. The customer wants to receive delivery within one month of signing of the contract. On the basis of the present production commitments, the production engineer forecasts that the plant has excess capacities of 90 hours of cutting time, 140 hours of forming machine time, 154 hours of welding time and 120 hours of finishing time.[9]

Input Data for Casing Production

Production design	Machine Time(mts.)				Total var. cost ($)
	Cutting	Forming	Welding	Finishing	
1	0.40	0.70	1.00	0.50	3.845
2	0.80	1.00	0.40	0.30	4.700
3	0.35	0.60	1.25	0.75	3.510
4	0.70	0.80	0.60	0.55	4.230

The above Linear Programming problem could be represented both in algebraic form and matrix form.

3. .1 Algebraic Form

$$x_1 + x_2 + x_3 + x_4 = 10{,}000 \quad \text{(requirement constraint)}$$

$$0.40x_1 + 0.80x_2 + 0.35x_3 + 0.70x_4 \leq 5400 \quad \text{(cutting time constraint)}$$

$$0.70x_1 + 1.00x_2 + 0.60x_3 + 0.80x_4 \leq 8400 \quad \text{(forming time constraint)}$$

$$1.00x_1 + 0.40x_2 + 1.25x_3 + 0.60x_4 \leq 9240 \quad \text{(welding time constraint)}$$

$$0.50x_1 + 0.30x_2 + 0.75x_3 + 0.55x_4 \leq 7200 \quad \text{(finishing time constraint)}$$

Objective Function:

$$5(x_1+x_2+x_3+x_4) - 3.845x_1 - 4.700x_2 - 3.51x_3 - 4.23x_4$$

$$\Rightarrow \quad \text{Max. } 1.155x_1 + 0.30x_2 + 1.49x_3 + 0.77x_4$$

Non-negativity Conditions:

$$x_j \geq 0, \qquad j = 1,2,3,4.$$

3.6.2 Matrix Form:

x_1	x_2	x_3	x_4		RHS
1.155	0.300	1.490	0.770		
1.00	1.00	1.00	1.00	=	10000
0.40	0.80	0.35	0.70	$\leq$	5400
0.70	1.00	0.60	0.80	$\leq$	8400
1.00	0.40	1.25	0.60	$\leq$	9240
0.50	0.30	0.75	0.55	$\leq$	7200

Slack and surplus variables should be created accordingly and coefficients of them should be incorporated into the matrix.

3.6.3 Formulation Using "LAMP":

Step (1): Declare MOLECULES; PRODUCTS, MACHINES

Step (2): Declare atoms under each molecule, i.e.

PRODUCTS <- PRODUCT1 , PRODUCT2 , PRODUCT3 , PRODUCT4

MACHINES <- CUTTING, FORMING, WELDING, FINISHING

Step (3): Enter the data by defining identifiers, i.e.,

(i) TIME.MACHINES.PRODUCTS $ (identifier)

(in a prompting to the system).

TIME.CUTTING.PRODUCT1 <- 0.40

TIME.CUTTING.PRODUCT2 <- 0.70

(ii) PROFIT.PRODUCT$ (identifier)

PROFIT.PRODUCT1 <- 1.155

PROFIT.PRODUCT2 <- 0.300

PROFIT.PRODUCT3 <- 1.490

PROFIT.PRODUCT4 <- 0.770

(Profit on each product = Sale price - Total variable cost)

(iii) TIME-AVLB.MACHINES$ (identifier)

TIME-AVLB.CUTTING <- 5400

TIME-AVLB.FORMING <- 8400

TIME-AVLB.WELDING <- 9240

TIME-AVLB.FINISHING <- 7200

(iv) REQUIREMEN$ (identifier)

REQUIREMEN <- 10000 (requirement)

Step (4): Declaration of Abstract Model:

```
SUM[PRODUCTS: TIME.MACHINES.PRODUCTS * PRODUCTS ?]
            <= TIME-AVLB.MACHINES    (constraints)
SUM[PRODUCTS: PRODUCTS ?] = REQUIREMEN
                                (requirement constraint)
MAX[PRODUCTS: PROFIT.PRODUCTS * PRODUCTS ?]
                                (objective function)
```

These declarations complete the formulation part in LAMP. The system interprets the declarations and solves the problem.

CHAPTER IV

CONCLUSIONS AND RECOMMENDATIONS

4.1 SUMMARY:

Mathematical Modeling is a potentially powerful tool in the Social Sciences and in Strategic Planning. Modeling of large-scale problems was quite costly and time-consuming process before the evolution of modern computers. With the development various computer systems, cost of numerical computations has been decreasing dramatically. Now the attention of system developers is focussed on providing a 'natural', 'expressable' and 'powerful' aid in developing models so that user and computer could be linked though they build/need the model in different forms.

These efforts have developed modeling languages which are very efficient and unambiguous in communicating models between humans as well as between humans and machines. An ideal modeling language should make the user to be able to communicate with machine in ordinary English language. Though this ideal seems to be beyond the reach of current day computer, tremendous efforts are being made to provide an English-like way of communicating.

4.2 ACHIEVEMENTS OF SYSTEM 'LAMP':

LAMP provides the user an English-like declaration for the constraints and objective function. This facilitates a good and proper communication between the modeler and persons interested in the model. The special feature of inputting the data from an external files makes large scale modeling easy. These files could be prepared by ordinary keypunch operators and a lot of modeler s time could be saved.

LAMP outputs the results in ordinary algebraic form. Thus making the output more understandable and appealing.

On the whole, LAMP bridges the gap between modeler s form (algebraic form) and machine-readable form (algorithmic form).

4.3 LIMITATIONS OF 'LAMP' AND EXTENSIONS POSSIBLE:

Though, LAMP provides an easy way of expressing the abstract model it is limited in some dimensions of the Linear Programming problem. LAMP could entertain a maximum of two indices for the coefficients, i.e. coefficients a_{ij} could be accepted. LAMP could be extended in this regard by incorporating a facility of accepting more indices. Though, generally Linear Programming problems do not encounter more than two-indexed coefficients but need may arise in some production planning and scheduling applications.

LAMP, though it can accept the constraint and objective function forms that generally appear in a Linear Programming problem, cannot accept complex expressions i.e. two or more expressions unitedly forming a constraint or objective functions. LAMP could be extended in this dimension also.

Another extension which makes LAMP more powerful is to hook it up with a Data Base Management System. Though LAMP can take data from external files but all the data required for a particular model should be available in that file. But the extension may make LAMP to access the specific data (required for the model) available on a large DATA BASE.

REFERENCES

1. Daniel R. Dolk and Benn R. Konsynski, Knowledge Representation for Model Management Systems, IEEE Tr. on Software Engg., Vol. SE-10, No. 6, Nov. 1984.

2. Fourer, R., Modeling Languages Versus Matrix Generators for Linear Programming, ACM Transactions on Mathematical Software, June 1983, Vol. 9, No. 2.

3. Fourer, R., and Harrison, M.J., A Modern Approach to Computer Systems for Linear Programming, Working Paper, Alfred P. Sloan School of Mgt., MIT, Cambridge (1978).

4. Johannes, B., and Meeraus, A., Towards a General Algebraic Modeling System, Development Research Centre, World Bank, 1978.

5. Katz, S., Risman, L.J., Rodeh, M., A System for Constructing Linear Programming Models, IBM Systems Journal, Vol. 19, No. 4, 1980.

6. Meeraus, A., An Algebraic Approach to Modeling, Proceedings on Economic Dynamics and Control, Denmark.

7. Michael Szu-Yuan Wang and James F. Courtney, Jr., A Conceptual Architecture for Generalised Decision Support System Software, IEEE Tr. on Systems, Man and Cybernetics, Vol. SMC-14, No. 5, Sept./Oct. 1984.

8. Ullman, J., Principles of Data Base Systems, Computer Science Press, (1980).

9. John. A. George and Hans, G. Dallenbach, Introduction to Operations Research Techniques, Allyn and Bacon (1978).